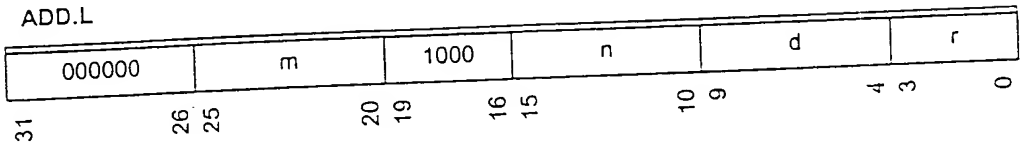
APPENDIX B

ADD.L

Description:

The ADD.L instruction adds the low 32 bits of R_m to the low 32 bits of R_n and stores the sign-extended 32-bit result in R_d. Bits $R_{m<32\ FOR\ 32>}$ and $R_{n<32\ FOR\ 32>}$ are ignored.

Operation:



ADD.L R_m, R_n, R_d

```
source1 ← SignExtend32(Rm);
source2 ← SignExtend32(Rn);
result ← SignExtend32(source1 + source2);
Rd ← Register(result);
```

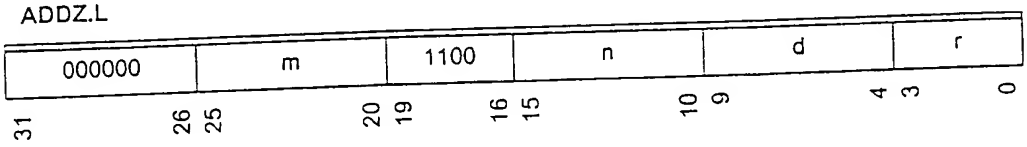
ADDZ.L

Description:

The ADDZ.L instruction adds the low 32 bits of R_m to the low 32 bits of R_n and stores the zero-extended 32-bit result in R_d. Bits $R_{m<32\ FOR\ 32>}$ and $R_{n<32\ FOR\ 32>}$ are ignored.

Operation:

ADDZ.L



ADDZ.L R_m, R_n, R_d

source1 ← $ZeroExtend_{32}(R_m)$;
source2 ← $ZeroExtend_{32}(R_n)$;
result ← $ZeroExtend_{32}$(source1 + source2);
R_d ← Register(result);

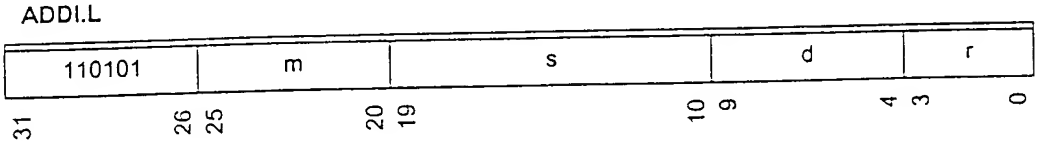
ADDI.L

Description:

The ADDI.L instruction adds the low 32 bits of R_m to the sign-extended value of the immediate *s*, and stores the sign-extended 32-bit result in the register R_d. Bits $R_{m<32\ FOR\ 32>}$ are ignored.

Operation:

ADDI.L



ADDI.L R_m, s, R_d

source1 ← $SignExtend_{32}(R_m)$;
source2 ← $SignExtend_{10}$(s);
result ← $SignExtend_{32}$(source1 + source2);
R_d ← Register(result);

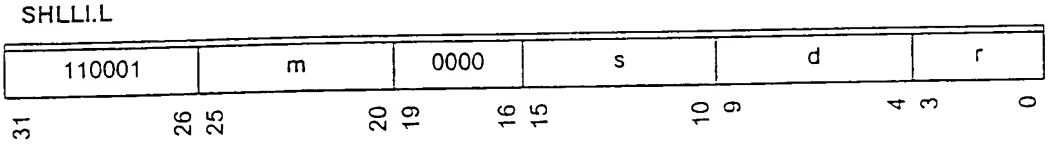
SHLLI.L

Description:

The SHLLI.L instruction logically left shifts the lower 32 bits of R_m by $s_{<0 \text{ FOR } 5>}$ and stores the sign-extended 32-bit result in R_d.

Operation:

SHLLI.L



SHLLI.L R_m, s, R_d

source1 ← ZeroExtend$_{32}$(R_m);
source2 ← ZeroExtend$_5$(SignExtend$_6$(s));
result ← SignExtend$_{32}$(source1 << source2);
R_d ← Register(result);

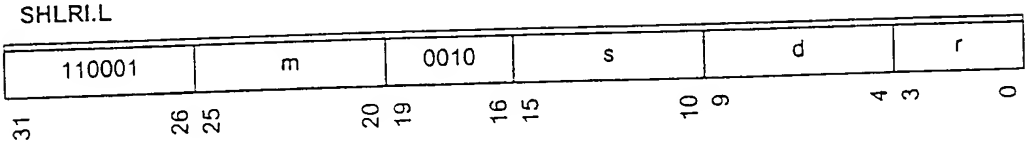
SHLRI.L

Description:

The SHLRI.L instruction logically right shifts the lower 32 bits of R_m by $s_{<0 \text{ FOR } 5>}$ and stores the sign-extended 32-bit result in R_d.

Operation:

SHLRI.L



SHLRI.L R_m, s, R_d

source1 ← $ZeroExtend_{32}(R_m)$;
source2 ← $ZeroExtend_5(SignExtend_6(s))$;
result ← $SignExtend_{32}$(source1 >> source2);
R_d ← Register(result);

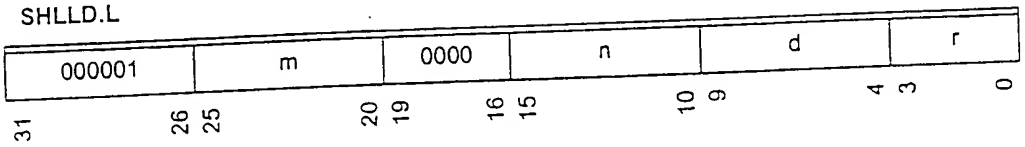
SHLLD.L

Description:

The SHLLD.L instruction logically left shifts the lower 32 bits of R_m by $R_{n<0 \text{ FOR } 5>}$ and stores the sign-extended 32-bit result in R_d.

Operation:

SHLLD.L



SHLLD.L R_m, R_n, R_d

source1 ← $ZeroExtend_{32}(R_m)$;
source2 ← $ZeroExtend_5(R_n)$;
result ← $SignExtend_{32}$(source1 << source2);
R_d ← Register(result);

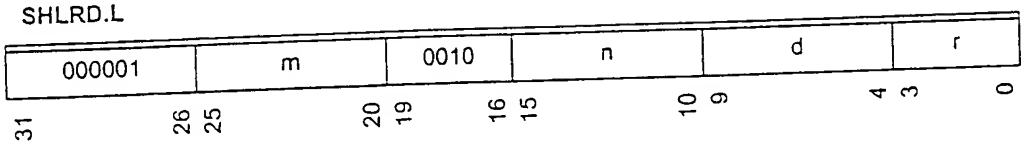
SHLRD.L

Description:

The SHLRD.L instruction logically right shifts the lower 32 bits of R_m by $R_{n<0\ FOR\ 5>}$ and stores the sign-extended 32-bit result in R_d.

Operation:

SHLRD.L



SHLRD.L R_m, R_n, R_d

```
source1 ← ZeroExtend32(Rm);
source2 ← ZeroExtend5(Rn);
result ← SignExtend32(source1 >> source2);
Rd ← Register(result);
```

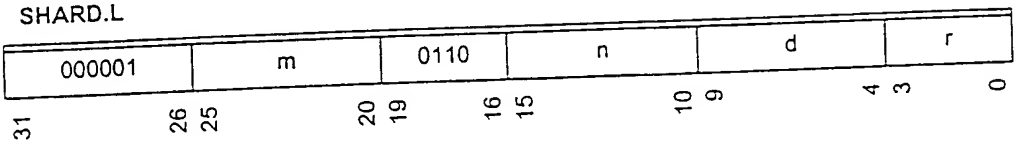
SHARD.L

Description:

The SHARD.L instruction arithmetically right shifts the lower 32 bits of R_m by $R_{n<0 \text{ FOR } 5>}$ and stores the sign-extended 32-bit result in R_d.

Operation:

SHARD.L



SHARD.L R_m, R_n, R_d

source1 ← $SignExtend_{32}(R_m)$;
source2 ← $ZeroExtend_5(R_n)$;
result ← $SignExtend_{32}$(source1 >> source2);
R_d ← Register(result);

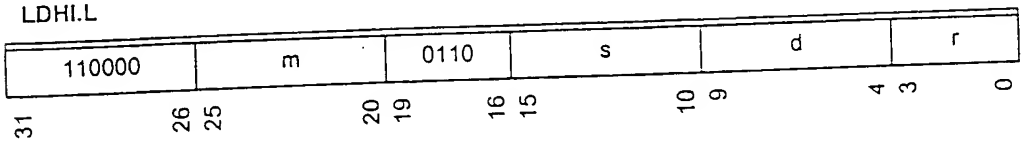
LDHI.L

Description:

Load the high part of a misaligned, signed long-word from memory to a general-purpose register.

Operation:

LDHI.L



LDHI.L R_m, s, R_d

```
base ← ZeroExtend64(Rm);
offset ← SignExtend6(s);
address ← base + offset;
count ← (address ∧ 0x3) + 1;
address ← ZeroExtend64(address ∧ (~ 0x3));
count8 ← count × 8;
shift ← ZeroExtend5((~ ((base + offset) ∧ 0x3)) × 8);
mem ← ZeroExtendcount8(MisalignedReadMemorycount8(address));
IF (IsLittleEndian())
   result ← SignExtend32(mem << shift);
ELSE
   result ← ZeroExtend32(mem);
Rd ← Register(result);
```

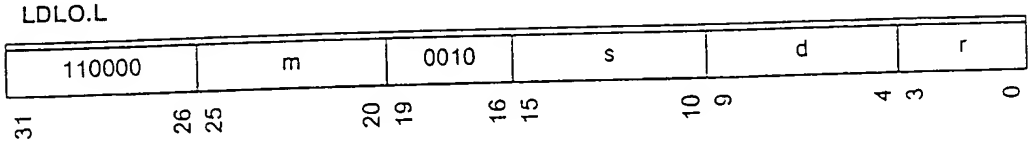
LDLO.L

Description:

Load the low part of a misaligned, signed long-word from memory to a general-purpose register.

Operation:

LDLO.L



LDLO.L R_m, s, R_d

```
base ← ZeroExtend64(Rm);
offset ← SignExtend6(s);
address ← ZeroExtend64(base + offset);
count ← 4 - (address ∧ 0x3);
count8 ← count × 8;
shift ← (address ∧ 0x3) × 8;
mem ← ZeroExtendcount8(MisalignedReadMemorycount8(address));
IF (IsLittleEndian())
   result ← ZeroExtend32(mem);
ELSE
   result ← SignExtend32(mem << shift);
Rd ← Register(result);
```

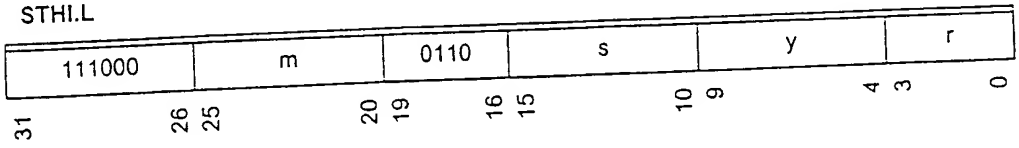
STHI.L

Description:

Misaligned store of the high part of a long-word from a general-purpose register to memory.

Operation:

STHI.L



STHI.L R_m, s, R_y

```
base ← ZeroExtend64(Rm);
offset ← SignExtend6(s);
value ← ZeroExtend32(Ry);
address ← base + offset;
count ← (address ∧ 0x3) + 1;
address ← ZeroExtend64(address ∧ (~ 0x3));
IF (IsLittleEndian())
   start ← (4 - count) × 8;
ELSE
   start ← 0;
count8 ← count × 8;
MisalignedWriteMemory count8(address, value< start FOR count8 >);
```

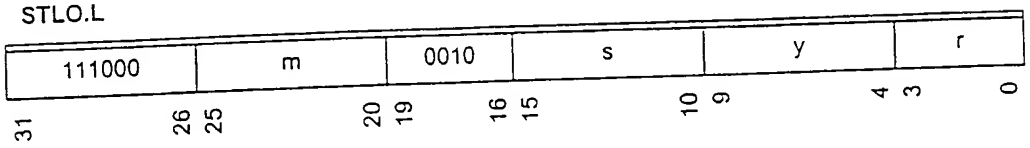
STLO.L

Description:

Misaligned store of the low part of a long-word from a general-purpose register to memory.

Operation:

STLO.L



STLO.L R_m, s, R_y

```
base ← ZeroExtend64(Rm);
offset ← SignExtend6(s);
value ← ZeroExtend32(Ry);
address ← ZeroExtend64(base + offset);
count ← 4 - (address ∧ 0x3);
IF (IsLittleEndian())
   start ← 0;
ELSE
   start ← (4 - count) × 8;
count8 ← count × 8;
MisalignedWriteMemory count8(address, value< start FOR count8 >);
```